

# The Online Replacement Path Problem

David Adjiashvili<sup>1</sup> and Marco Senatore<sup>2</sup>

<sup>1</sup> Institute for Operations Research (IFOR)  
Eidgenössische Technische Hochschule (ETH) Zürich  
Rämistrasse 101, 8092 Zürich, Switzerland  
`david.adjiashvili@ifor.math.ethz.ch`

<sup>2</sup> Dipartimento di Informatica, Sistemi e Produzione  
Università di Roma Tor Vergata  
Via del Politecnico 1, 00133 Rome, Italy  
`senatore@disp.uniroma2.it`

**Abstract.** We study a natural online variant of the replacement path problem. The *replacement path problem* asks to find for a given graph  $G = (V, E)$ , two designated vertices  $s, t \in V$  and a shortest  $s$ - $t$  path  $P$  in  $G$ , a *replacement path*  $P_e$  for every edge  $e$  on the path  $P$ . The replacement path  $P_e$  is simply a shortest  $s$ - $t$  path in the graph, which avoids the *failed* edge  $e$ . We adapt this problem to deal with the natural scenario, that the edge which failed is not known at the time of solution implementation. Instead, our problem assumes that the identity of the failed edge only becomes available when the routing mechanism tries to cross the edge. This situation is motivated by applications in distributed networks, where information about recent changes in the network is only stored locally, and fault-tolerant optimization, where an adversary tries to delay the discovery of the materialized scenario as much as possible. Consequently, we define the *online replacement path problem*, which asks to find a nominal  $s$ - $t$  path  $Q$  and detours  $Q_e$  for every edge on the path  $Q$ , such that the worst-case arrival time at the destination is minimized. Our main contribution is a label setting algorithm, which solves the problem in undirected graphs in time  $O(m \log n)$  and linear space for all sources and a single destination. We also present algorithms for extensions of the model to any bounded number of failed edges.

## 1 Introduction

Modeling the effects of limited reliability of networks in modern routing schemes is important from the point of view of most applications. It is often unrealistic to assume that the nominal network known at the stage of decision making will be available in its entirety at the stage of solution implementation. Several research directions have emerged as a result. The main paradigm in most works is to obtain a certain 'fault-tolerant' or 'redundant' solution, which takes into account a certain set of likely network realizations at the implementation phase. One important example is the *replacement path problem* (RP) [18,11]. The input in RP is a nominal network given as a graph  $G = (V, E)$ , a source  $s$ , a destination

$t$  and one shortest  $s$ - $t$  path  $P$ . The goal is to find for every edge  $e$  on the path  $P$ , a shortest path in  $G$ , which does not use the edge  $e$ . RP attempts to model the situation in which any link in the network may fail before the routing process starts. It is hence desirable to compute in advance the shortest replacement paths, for the case of a failure of any one of the edges in the nominal path  $P$ . In the event of a failure the routing mechanism simply chooses the corresponding pre-computed path. The applicability of RP is, however, limited to those situations, in which it is possible to know the identity of the failed link before the routing process starts. This assumption is not realistic in many important applications, in which faults in the network occur 'online', or the information about them is stored in a distributed fashion. The latter situation is commonplace, for example, in transportation networks (e.g. accidents in road networks). Furthermore, it is a common feature of very large networks, such as the Internet. This paper studies the *online replacement path problem* (ORP), which captures this online failure setting. The most notable difference between RP and ORP is that in ORP we assume that the routing mechanism is informed about the failed link at the moment it tries to use it. Another important difference is related to the nominal path  $P$ . In RP a certain nominal shortest path is provided in the input. This is no longer the case in ORP, since the detour taken in the event of a failure does not always start from the source vertex  $s$ . This means that in ORP we simultaneously optimize both the nominal path and the optimal detours, taking into account a certain global objective function. An informal formulation of ORP is as follows. We would like to route a certain package through a network from a given source to a given destination as quickly as possible. We are aware of the existence of a failed link in the network, but we do not know its location. It is possible to observe that a certain link has failed by *probing* it. In order to probe a link, the package should be at one of the endpoints of this link. If a probed link is intact, the package crosses the link to the other endpoint and the cost of traversing the link is incurred. Otherwise the package stays in the same endpoint and the routing mechanism is informed about the failed link. In other words, it is only possible to observe that a link has failed by trying to cross it. The goal is to find a set of paths (a nominal path and detours for every edge on the nominal path) that will minimize the latest possible arrival time to the destination. A solution of ORP should hence specify both the nominal path and the optimal detours at every vertex along the path, which avoid the next edge on the nominal path.

The latter informal definition suggests that in ORP we take a conservative fault-tolerant approach. In fact, it is assumed that a failed link does exist in the network, but its identity is unknown. This suggests another application of ORP. In many applications it is only necessary to route a certain object within a certain time, called a *deadline*. As long as the object reaches its destination before the deadline, no penalty is incurred. On the other hand, if the deadline is not met, a large penalty is due. An example of such an application is organ transportation for transplants (see e.g. Moreno, Valls and Ribes [14]), in which it is critical to deliver a certain organ before the scheduled time for the surgery. In

this application it does not matter how early the organ arrives at the destination, as long as it arrives in time. In such applications it is often too risky to take an unreliable shortest path, which admits only long detours in some scenarios, whereas a slightly longer path with reasonably short detours meets the deadline in *every* scenario. Hence, with ORP it is often possible to immunize the path against faults in the network.

The main result of this paper is that the solution to ORP in undirected networks can be computed in  $O(m \log n)$  time and linear space for all sources and a single destination, where  $n$  and  $m$  are the number of vertices and edges in the network, respectively. Furthermore, this solution can be stored in  $O(n)$  space. The basic algorithm is a label-setting algorithm, similar to Dijkstra’s algorithm for ordinary shortest paths. We describe this algorithm in Section 3. The main technical difficulty lies in the need to pre-compute certain shortest path distances in modified graphs. This difficulty is overcome in Section 4, which provides a fast implementation of this step. We generalize the model to incorporate the possibility of an arbitrary bounded number  $k$  of failed edges in Section 5. This section also gives an alternative view on the problem using the notion of *routing strategies*, and provides a polynomial algorithm for the problem. We mention other results linking ORP with the shortest path problem in Section 6. In particular, we show that it is possible to solve a bi-objective variant of ORP. This result implies that it is possible to efficiently obtain Pareto-optimal paths with respect to ordinary distance and the cost corresponding to the ORP problem, making it an attractive method for various applications. We also analyze the performance of a greedy heuristic which attempts to route along a shortest path in the remaining network. We show that this heuristic, which is implemented in many applications, is a poor approximation for ORP. We summarize in Section 7. The following section reviews related work.

## 2 Related Work

The replacement path problem was first introduced by Nisan and Ronen [18]. The motivation for their definition stemmed from the following question in auction theory: what is the true price of a link in a network, when we try to connect two distinct vertices  $x$  and  $y$ , and every edge is owned by a self-interested agent. It turns out that compensating agents with respect to the declarations of other agents in the auction leads to truthful declarations, namely declarations which reflect the true costs of the agent. Such pricing schemes are called *Vickrey schemes*. In the setup of networks, replacement paths lengths correspond to Vickrey prices for the individual failed edges. Another important application of RP is the *k shortest simple paths problem* (kSSP), which reduces to  $k$  replacement path computations.

The complexity of the RP problem for undirected graphs is well understood. Malik, Mittal and Gupta [13] give a simple  $O(m+n \log n)$  algorithm. A mistake in this paper was later corrected by Bar-Noy, Khuller and Schieber [4]. This running time is asymptotically the same as a single source shortest path computation.

Nardelli, Proietti and Widmayer [17] later provided an algorithm with the same complexity for the variant of RP, in which vertices are removed instead of edges. The same authors give efficient algorithms for finding detour-critical edges for a given shortest path in [15,16].

In directed graphs the situation is significantly different. A trivial upper bound for RP corresponds to  $O(n)$  single shortest path computations. This gives  $O(n(m + n \log n))$  for general directed graphs with nonnegative weights. This was slightly improved to  $O(mn + n^2 \log \log n)$  by Gotthilf and Lewenstein [10]. The challenge of improving the  $O(mn)$  bound for RP on directed graphs was mainly tackled by restricting the class of graphs or by allowing approximate solutions. Along the lines of the former approach, algorithms were developed for unweighted graphs (Roditty and Zwick [21]) and planar graphs (Emek, Peleg and Roditty [7], Klein, Mozes, and Weimann [12] and Wulff-Nilsen [23]). The latter approach was successfully applied to obtain  $\frac{3}{2}$ -approximate solutions by Roditty [20] and  $(1 + \epsilon)$ -approximate solutions by Bernstein [5]. Weimann and Yuster [22] applied fast matrix multiplication techniques to obtain a randomized algorithm with sub-cubic running time for certain ranges of the edge weights.

Another problem which bears resemblance to ORP is the *stochastic shortest path with recourse problem* (SSPR), studied by Andreatta and Romeo [3]. This problem can be seen as the stochastic analogue of ORP.

Finally, we briefly review some related work on robust counterparts of the shortest path problem. The shortest path problem with cost uncertainty was studied by Yu and Yang [24], who consider several models for the scenario set. These results were later extended by Aissi, Bazgan and Vanderpooten [2]. These works also considered a two-stage min-max regret criterion. Dhamdhare, Goyal, Ravi and Singh [6] developed the demand-robust model and gave an approximation algorithm for the shortest path problem. A two-stage feasibility counterpart of the shortest path problem was addressed in Adjashvili and Zenklusen [1]. Puhl [19] provided hardness results for numerous two-stage counterparts the shortest path problem, and gave some approximation algorithms.

### 3 An Algorithm for ORP

In this section we develop an algorithm for ORP. Some technical proofs are left in Appendix A. Let us establish some notations first. We are given an undirected edge-weighted graph  $G = (V, E, \ell)$ , a source  $s \in V$  and destination  $t \in V$ . We are assuming throughout this paper that the edge weights  $\ell$  are nonnegative. For two vertices  $u, v \in V$  let  $\mathcal{P}_{u,v}$  denote the set of simple  $u$ - $v$  paths in  $G$ . Let  $N(u)$  denote the set of neighbors of  $u$  in  $G$ . For a set of edges  $A \subset E$  let  $\ell(A) = \sum_{e \in A} \ell(e)$ . For an edge  $e \in E$  and a set of edges  $F \subseteq E$ , let  $G - e$  and  $G - F$  denote the graph obtained by removing the edge  $e$  and the edges in  $F$ , respectively. For a graph  $H$  let  $d_H(\cdot, \cdot)$  denote the shortest path distance in  $H$ . Paths are always represented as sets of edges, while walks are represented as sequences of vertices. For a path  $P$  with incident vertices  $u$  and  $v$  let  $P[u, v]$

denote the subpath of  $P$  from  $u$  to  $v$ . For an edge  $e \in E$  and  $u \in V$  let

$$s_u^{-e} = d_{G-e}(u, t)$$

denote the shortest  $u$ - $t$  path distance in  $G - e$ .

Our algorithm uses a label-setting approach, analogous to Dijkstra's algorithm for shortest paths. In other words, in every iteration the algorithm updates certain tentative labels for the vertices of the graph, and fixes a final label to a single vertex  $u$ . This final label represents the connection cost of  $u$  by an optimal path to  $t$ .

**Definition 1.** *Given a vertex  $v \in V$ , the robust length of the  $v$ - $t$  path  $P$  is*

$$\text{Val}(P) = \max\{\ell(P), \max_{uu' \in P} \{\ell(P[v, u]) + s_u^{-uu'}\}\}.$$

*The potential  $y(v)$  is defined as the minimum of  $\text{Val}(P)$  over all  $P \in \mathcal{P}_{v,t}$ , and any path  $P^*$  attaining  $\text{Val}(P^*) = y(v)$  is called an optimal nominal path. Finally, ORP is to compute  $y(s)$  and obtain a corresponding optimal nominal path.*

The robust length of a  $v$ - $t$  path  $P$  is simply the maximal possible cost incurred by following  $P$  until a certain vertex, and then taking the best possible detour from that vertex to  $t$  which avoids the next edge on the path. To avoid confusion, we stress that in ORP we assume the existence of at most one failed edge in the graph. Consider next a scenario in which an edge  $uu' \in P$  fails and let  $u \in V$  be the vertex which is closer to  $v$ . We can assume without loss of generality that the best detour is a shortest  $u$ - $t$  path in the graph  $G - uu'$ . Critically, the values  $s_u^{-uu'}$  are independent of the chosen path.

Observe that from non-negativity of  $\ell$  we obtain  $\text{Val}(P) \geq \text{Val}(P')$ , whenever  $P$  and  $P'$  are  $u$ - $t$  and  $v$ - $t$  paths respectively, and  $P'$  is a subpath of  $P$ . We denote this property by *monotonicity*. Furthermore, we can prove the following.

**Lemma 1.** *Let  $P_u \in \mathcal{P}_{u,t}$  and let  $v \in N(u)$  be a vertex, not incident to  $P_u$ . Then the path  $P_v = P_u \cup \{vu\}$  satisfies*

$$\text{Val}(P_v) = \max\{\ell(vu) + \text{Val}(P_u), s_v^{-vu}\}.$$

Our algorithm for ORP updates the potential on the vertices of the graph, using the property established by the following lemma.

**Lemma 2.** *Let  $U \subset V$ , with  $t \in U$ , be the set of vertices for which the potential is known. And let  $uv$  be the edge such that:*

$$uv = \arg \min_{zw \in E: w \in U, z \in V \setminus U} \{\max\{\ell(zw) + y(w), s_z^{-zw}\}\}. \quad (1)$$

*Then if  $P_u \in \mathcal{P}_{u,t}$  with  $\text{Val}(P_u) = y(u)$  and  $P_v = P_u \cup \{uv\}$  it holds that  $\text{Val}(P_v) = y(v)$ .*

---

**Algorithm 1**

---

```
1: Compute  $s_u^{-uv}$  for each  $uv \in E$ .
2:  $U = \emptyset$ ;  $W = V$ ;  $y'(t) = 0$ ;  $y'(u) = \infty \forall u \in V - t$ .
3:  $successor(u) = \text{NIL} \forall u \in V$ .
4: while  $U \neq V$  do
5:   Find  $u = \arg \min_{z \in W} y'(z)$ .
6:    $U = U + u$ ;  $W = W - u$ ;  $y(u) = y'(u)$ .
7:   for all  $vu \in E$  with  $v \in W$  do
8:     if  $y'(v) > \max\{\ell(vu) + y'(u), s_v^{-vu}\}$  then
9:        $y'(v) = \max\{\ell(vu) + y'(u), s_v^{-vu}\}$ .
10:     $successor(v) = u$ .
```

---

Lemma 2 provides the required equation for our label-setting algorithm, whose formal statement is given as Algorithm 1. The algorithm iteratively builds up a set  $U$ , consisting of all vertices, for which the correct potential value of  $y(u)$  was already computed. The correctness of the algorithm is a direct consequence of Lemma 2.

Consider the running time of Algorithm 1. We let  $n$  and  $m$  denote the number of vertices and edges of the input graph, respectively. An efficient implementation of step 1 is delayed to the next section, and constitutes the heart of our efficient algorithms. For steps 2-10 we use the implementation of Fredman and Tarjan [8] for priority queues (heaps) called *Fibonacci Heaps*. We adopt here the same implementation that is used to obtain  $O(m + n \log n)$  running time for Dijkstra's algorithm. We omit the details as they are identical to those in [8]. We comment that our implementation of step 1 uses  $O(m \log n)$  time, hence this computation dominates the running time. In fact, the running time of Algorithm 1 remains the same if steps 2-10 are implemented using simpler data structures, such as Binary Heaps.

Finally, let us remark that Algorithm 1 works both for directed and undirected graphs. However, the following section provides an implementation of step 1 for undirected graphs only. We remark that in directed graphs, Algorithm 1 can be trivially implemented in time  $O(n \text{SP}(n, m))$ , where  $\text{SP}(n, m)$  is the complexity of a single shortest path computation on a graph with  $n$  vertices and  $m$  edges. This is a simple consequence of our first observation in the following section.

## 4 Efficient Computation of $s$ -Values

It remains to provide an efficient implementation for the computation of the values  $s_u^{-uu'}$ . We will assume a random access machine (RAM) as a computational model.

We start by computing the shortest path tree  $T$  in  $O(m \log n)$  time from every vertex to  $t$ . Let  $d^*(u) = d_G(u, t)$ . Observe that  $s_u^{-uu'} = d^*(u)$  holds for every edge  $uu'$  outside of  $T$ . We can hence concentrate our efforts on computing

$s$ -values for edges in the tree. For a vertex  $v \in V$  we denote by  $T_v \subset T$  the subtree rooted at  $v$ . We set  $E' = E \setminus T$ .

**Lemma 3.** *Consider a vertex  $u \in V$ . Let  $vw \in E'$  be an edge attaining*

$$\min_{vw \in E', v \in T_u, w \notin T_u} \{d_G(u, v) + \ell(vw) + d^*(w)\}, \quad (2)$$

*and let  $uu'$  be the first edge on the  $u$ - $t$  path in  $T$ . Then*

$$s_u^{-uu'} = d_G(u, v) + \ell(vw) + d^*(w).$$

By defining  $c_{vw} = d^*(v) + d^*(w) + \ell(vw)$  for each edge  $vw \in E'$  and substituting in the expression for  $s_u^{-uu'}$  obtained in the previous lemma we can write

$$s_u^{-uu'} = c_{vw} - d^*(u), \quad (3)$$

which is a convenient expression for computing the  $s$ -values. Indeed this expression shows that value  $s_u^{-uu'}$  only depends on the lowest value  $c_e$ , over all  $e$  with exactly one incident vertex in  $T_u$ .

We leave the remaining details of our implementation to the proof of Theorem 1. Before stating the theorem, let us define an important ingredient, which is used hereafter. To efficiently obtain the information, whether a certain edge  $e \in E'$  corresponds to a feasible detour for  $u$  (namely if exactly one endpoint of  $e$  is in  $T_u$ ), we need an algorithm for computing the *least common ancestor* (LCA) in trees. Given a tree rooted at a vertex  $t$  and two vertices  $u, v$  in the tree, the least common ancestor  $\text{lca}(u, v)$  of  $u$  and  $v$  is the vertex at which the  $t$ - $u$  and  $t$ - $v$  paths diverge in the tree. For an edge  $e$  we write  $\text{lca}(e)$  to denote the LCA of the endpoints of  $e$ . It is straightforward to see that an edge  $e = vw$  corresponds to a feasible detour for an edge  $uu'$  if and only if either the  $v$ - $\text{lca}(v, w)$  path or the  $w$ - $\text{lca}(v, w)$  path in  $T$  contain  $uu'$ . For this purpose we use the algorithm of Gabow and Tarjan [9], which uses  $O(n + p)$  time to compute the LCA of  $p$  pairs of vertices in a rooted tree. In our case we have  $p = O(m)$ , since we would like to compute the LCA for every pair of vertices connected by an edge  $e \in E'$ , hence this computation takes  $O(m)$  time. We assume henceforth that given an edge  $e \in E'$  we have access to  $\text{lca}(e)$  in constant time.

**Theorem 1.** *The values  $s_u^{-uu'}$  can be computed in  $O(m \log n)$  time and linear space for all  $uu' \in E$ . Furthermore, they can be stored in a data structure of size  $O(n)$ .*

*Proof.* The algorithm is summarized as Algorithm 2 in Appendix B. The algorithm starts by sorting the set of edges  $E'$  according to increasing order of  $c_e$ . Let  $L$  be the sorted list. This operation (as well as the computation of the values  $c_e$ ) takes  $O(m \log n)$  time.

The algorithm relies on the following fact. Let  $e$  be the first edge in the sorted list  $L$ , which represents a feasible detour for  $uu' \in T$ . From the previous discussion, the edge  $e$  corresponds to the optimal detour for  $uu'$ . Furthermore, if

we traverse the two paths from the endpoints of  $e$  to  $\text{lca}(e)$  we cross the edge  $uu'$ . The algorithm indeed performs the latter traversals in a copy of  $T$  and marks all edges along the way (which were not marked before, using an edge that has lower  $c$ -value) as belonging to the edge  $e$ . To obtain the desired running time we need to avoid traversing the same parts of  $T$  several times. To achieve this we create a second copy  $\bar{T}$  of the tree  $T$ . For every vertex  $u \in T$  we store a pointer  $p[u]$  from  $u \in T$  to its copy  $\bar{u}$  in  $\bar{T}$ , and another pointer  $p[\bar{u}]$ , pointing in the opposite direction. We start iterating over the edges in the sorted list  $L$ . In the beginning of the  $i$ 'th iteration, the first edge  $e_i = xy$  in  $L$  is removed from  $L$ , and the algorithm jumps to the corresponding vertices  $\bar{x} = p[x]$  and  $\bar{y} = p[y]$  in  $\bar{T}$ . The algorithm also memorizes  $\bar{z} = \text{lca}(\bar{x}, \bar{y})$  - the vertex in  $\bar{T}$ , corresponding to  $z = \text{lca}(x, y)$ . Next the  $\bar{x}-\bar{z}$  and  $\bar{y}-\bar{z}$  paths are traversed in  $\bar{T}$ , marking all edges along the way as belonging to  $e_i$ , and computing the  $s$ -values for them using (3). Finally all these aforementioned edges are contracted in  $\bar{T}$  and the pointers are updated so that every vertex in  $T$  always points to its corresponding super-vertex in  $\bar{T}$  and vice-versa. Finally, the list is post-processed to remove some unnecessary edges in the following way. At every step, the first edge  $e = vw$  in the list is inspected. If  $p[v] = p[w]$ , this edge is a self-loop in  $\bar{T}$ , and it can not correspond to correct  $s$ -value assignments anymore. Consequently, this edge is removed from the list and the next edge is inspected. The process ends when  $p[v] \neq p[w]$ , or when  $L$  is empty. In the latter case the algorithm terminates. This concludes the  $i$ 'th iteration.

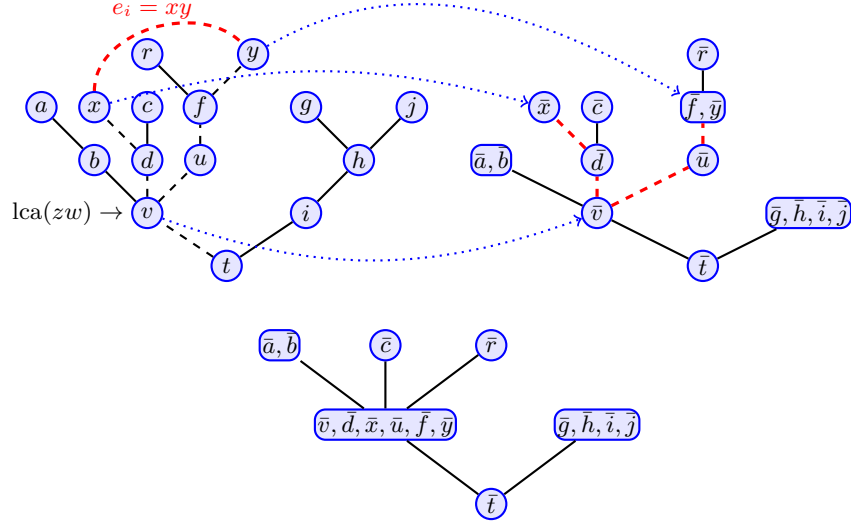
Note that the running time of the  $i$ 'th iteration is proportional to the combined length of the  $\bar{x}-\bar{z}$  path, the  $\bar{y}-\bar{z}$  path and the prefix of  $L$  that is deleted in the post-processing. Since we contract every edge we traverse, in the tree  $\bar{T}$ , we never traverse the same edge twice. We conclude that the running time of this last stage of the algorithm is  $O(n + m)$ . Figure 1 illustrates the algorithm.

Finally, note that we can store a representation of all  $s$ -values and the corresponding detours in  $O(n)$  space. This is achieved by storing the tree  $T$ , and for every vertex  $u$  and the corresponding edge  $uu'$  in  $T$ , we store a pointer to the edge attaining the minimum in (2). The values  $s_u^{-uu'}$  can either be stored in a separate list using  $O(m)$  additional space, or computed in constant time from the aforementioned information, by using (3). □

Note that the running time of Algorithm 2 is dominated by the sorting of the costs of the edges. This fact is a significant advantage since sorting algorithms are very efficient in practice. In fact, if the sorted list  $L$  was provided in the input, the running time of the algorithm could be improved to  $O(m + n \log n)$ , via the implementation mentioned in Section 3. Another case in which the latter complexity bound can be attained is that of unweighted graphs, where bucketing can be used in order to sort the costs  $c_e$  in linear time. Theorem 2 summarizes our main result.

**Theorem 2.** *Given an instance of ORP the potential  $y$  and the corresponding paths can be computed in time  $O(m \log n)$ .*





**Fig. 1.** Top left: The shortest path tree  $T$ . The next edge in  $E'$  to be processed is  $e_i = xy$ . The cost  $c_{xy}$  associated with this edge is the sum of weights of all dashed edges, with the edge  $vt$  counted twice.  $\text{lca}(xy) = v$ . Top right: The tree  $\bar{T}$  in the beginning of the  $i$ 'th iteration. The labels of the super-vertices correspond to the sets of vertices in  $T$  associated with them. The dashed edges are traversed in the  $i$ 'th iteration and contracted to form the tree on the bottom of the figure.

## 5 $k$ -ORP

Let us formally define  $k$ -ORP, the online replacement path problem with  $k$  failed edges. We refer to the parameter  $k$  as the *failure parameter*. In  $k$ -ORP a *scenario* corresponds to a removal of any  $k$  of the edges in the graph. In this setup it is no longer convenient to describe the problem in terms of paths and detours. Instead we introduce the notion of a *routing strategy*.

A routing strategy  $R : 2^E \times V \rightarrow V$  is a function which, given a subset  $F' \subset E$  of known failed links and a vertex  $v \in V$ , returns a vertex  $u \in V$ . We call  $E' = E \setminus F'$  the set of *active edges*. We are assuming the existence of a certain governing mechanism, which takes as input a routing strategy and executes it on a given instance. Provided with a routing strategy  $R$ , this mechanism iteratively moves from the current vertex  $u$  to the vertex  $v = R(F', u)$ , where  $F'$  is the set of failed links probed so far. The process starts at a given origin  $s$  with  $F' = \emptyset$  and ends when  $t$  is reached.

Since  $R$  is deterministic, this process defines a unique, possibly infinite, walk  $\theta_R(u, E, F)$  in  $G$  for each origin  $u \in V$  and every scenario  $F \subset E$  with  $|F| \leq k$ . We remark that if  $G$  contains at least  $k + 1$  edge-disjoint  $s$ - $t$  paths, there exists a routing strategy, which does not cycle.

**Definition 2.** Given  $E' \subset E$ , a vertex  $u \in V$  and a routing strategy  $R$ , the  $k$ -value of  $R$  with respect to  $E'$  and  $u$  is defined as

$$\text{Val}_k(u, E', R) = \max_{F \subseteq E', |F| \leq k} \ell(\theta_R(u, E', F)).$$

The corresponding  $k$ -potential of  $u$  with respect to  $E'$  is defined as

$$y^k(u, E') = \min_R \text{Val}_k(u, E', R).$$

Finally,  $k$ -ORP is to find an *optimal routing strategy*  $R^*$ , which minimizes  $\text{Val}_k(s, E, R)$ , namely to solve

$$R^* = \arg \min_R \text{Val}_k(s, E, R).$$

The following relation, which is a generalization of (1), gives rise to a simple recursive algorithm for  $k$ -ORP. For each  $v \in V$  it holds:

$$y^k(v, E) = \min_{v:vu \in E} \{\max\{\ell(vu) + y^k(u, E), y^{k-1}(v, E \setminus vu)\}\}. \quad (4)$$

The correctness of this relation can be proved by induction on  $k$ , using the arguments in Section 3. The exact statement of the algorithm is given in Appendix C, alongside a complexity analysis. We summarize with the following theorem.

**Theorem 3.**  $k$ -ORP can be solved on undirected graphs in time  $O(m^k \log n)$ .

## 6 ORP vs. Shortest Paths

As we have seen, the length of a path and its robust length are different, and often conflicting objectives functions. In this section we mention two results relating these two objectives.

Consider first the problem of finding an optimal solution to an instance of ORP with the shortest possible nominal path length. This bi-objective problem asks to find a Pareto-optimal path with respect to the latter two objective functions. Our first result asserts that this problem can be solved in polynomial time with a simple adaptation of Dijkstra's algorithm. In fact, we are able to solve the following problem for every bound  $B \geq OPT$ .

$$P^* = \arg \min_{P \in \mathcal{P}_{s,t}: \text{Val}(P) \leq B} l(P).$$

The algorithm and possible applications of this problem are given in Appendix D.

The second result analyzes the performance of certain greedy heuristics for  $k$ -ORP. We show that a routing strategy that always tries to route along the shortest path in the remaining graph is a  $(2^{k+1} - 1)$ -approximation algorithm for  $k$ -ORP. The details are available in Appendix E.

## 7 Conclusions

This paper introduces a natural variant of the replacement path problem, the online replacement path problem. ORP captures many real-life situations, which occur in faulty or large distributed networks. The most important characteristic of ORP is that the information about the failed edge in the network is only available locally, namely on the edge itself. ORP possesses many of the nice characteristics of ordinary shortest paths. In particular, we show that ORP can be solved by a simple label-setting algorithm. The computational bottleneck of this algorithm is the need to pre-compute certain shortest paths in an adapted network. We give an efficient implementation of this step, which uses  $O(m \log n)$  time and linear space. We also generalize the algorithm to deal with an arbitrary constant number  $k$  of failed edges. Along this vein we introduce the notion of a routing strategy. Finally, we observe that a Pareto-optimal path with respect to ordinary distance and robust length can be found in polynomial time.

We conclude by mentioning a number of promising directions for future research. The complexity of ORP with a variable number  $k$  of failed edges remains open. In fact, it is not clear if the decision problem  $y^k(s, E) \leq M$  is in NP. The complexity of ORP in undirected graphs may potentially be improved to  $O(m + n \log n)$ . Finally, the complexity of ORP in directed graphs remains open. Our results give an algorithm with the running time of  $O(n \text{SP}(n, m))$ , where  $\text{SP}(n, m)$  is the complexity of a single shortest path computation on a graph with  $n$  vertices and  $m$  edges. This problem threatens to be as challenging as RP in directed graphs.

## References

1. D. Adjiashvili and R. Zenklusen. An  $s - t$  connection problem with adaptability. *Discrete Applied Mathematics*, 159(8):695 – 705, 2011.
2. H. Aissi and D. Bazgan, C. Vanderpooten. Approximation complexity of min-max (regret) versions of shortest path, spanning tree, and knapsack. In *ESA*, pages 862–873, 2005.
3. G. Andreatta and L. Romeo. Stochastic shortest paths with recourse. *Networks*, 18(3):193–204, 1988.
4. A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical report, Univ. of Maryland Institute for Advanced Computer Studies Report No. UMIACS-TR-95-96, College Park, MD, USA, 1995.
5. A. Bernstein. A nearly optimal algorithm for approximating replacement paths and  $k$  shortest simple paths in general graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 742–755, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.
6. K. Dhamdhere, V. Goyal, R. Ravi, and M. Singh. How to pay, come what may: Approximation algorithms for demand-robust covering problems. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 367–378, Washington, DC, USA, 2005. IEEE Computer Society.
7. Y. Emek, D. Peleg, and L. Roditty. A near-linear-time algorithm for computing replacement paths in planar directed graphs. *ACM Trans. Algorithms*, 6:64:1–64:13, September 2010.

8. M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, July 1987.
9. H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 246–251, New York, NY, USA, 1983. ACM.
10. Z. Gotthilf and M. Lewenstein. Improved algorithms for the  $k$  simple shortest paths and the replacement paths problems. *Inf. Process. Lett.*, 109:352–355, March 2009.
11. J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proceedings of the 42nd Symposium on the Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, pages 252–259, 2001.
12. P. N. Klein, S. Mozes, and O. Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space  $o(n \log^2 n)$ -time algorithm. *ACM Trans. Algorithms*, 6:30:1–30:18, April 2010.
13. K. Malik, A. K. Mittal, and S. K. Gupta. The  $k$  most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, August 1989.
14. A. Moreno, A. Valls, and A. Ribes. Finding efficient organ transport routes using multi-agent systems. In *Proceedings of the IEEE 3rd International Workshop on Enterprise Networking and Computing in Health Care Industry (Healthcom)*, pages 233–258, 2001.
15. E. Nardelli, G. Proietti, and P. Widmayer. Finding the detour-critical edge of a shortest path between two nodes. *Information Processing Letters*, 67(1):51–54, 1998.
16. E. Nardelli, G. Proietti, and P. Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.
17. E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. In Jie Wang, editor, *Computing and Combinatorics*, volume 2108 of *Lecture Notes in Computer Science*, pages 278–287. Springer Berlin / Heidelberg, 2001.
18. N. Nisan and A. Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 129–140, New York, NY, USA, 1999. ACM.
19. C. Puhl. Recoverable robust shortest path problems. Preprint 034-2008, Institute of Mathematics, Technische Universität Berlin, 2008.
20. L. Roditty. On the  $k$ -simple shortest paths problem in weighted directed graphs. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 920–928, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
21. L. Roditty and U. Zwick. Replacement paths and  $k$  simple shortest paths in unweighted directed graphs. In *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 100–100. Springer Berlin / Heidelberg, 2005.
22. O. Weimann and R. Yuster. Replacement paths via fast matrix multiplication. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:655–662, 2010.
23. C. Wulff-Nilsen. Solving the replacement paths problem for planar directed graphs in  $o(n \log n)$  time. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 756–765, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.
24. G. Yu and J. Yang. On the robust shortest path problem. *Computers & Operations Research*, 25(6):457 – 468, 1998.

## A Proofs

### A.1 Proof of Lemma 1

Applying the definition we compute

$$\begin{aligned}\ell(vu) + \text{Val}(P_u) &= \ell(vu) + \max\{\ell(P_u), \max_{zw \in P_u} \{\ell(P_u[u, z]) + s_z^{-zw}\}\} \\ &= \max\{\ell(vu) + \ell(P_u), \max_{zw \in P_u} \{\ell(vu) + \ell(P_u[u, z]) + s_z^{-zw}\}\} \\ &= \max\{\ell(P_v), \max_{zw \in P_v \setminus \{vu\}} \{\ell(P_v[v, z]) + s_z^{-zw}\}\}.\end{aligned}$$

Substituting in the desired expression we obtain

$$\begin{aligned}\max\{\ell(vu) + \text{Val}(P_u), s_v^{-vu}\} &= \\ \max\{\ell(P_v), \max_{zw \in P_v \setminus \{vu\}} \{\ell(P_v[u, z]) + s_z^{-zw}\}, s_v^{-vu}\} &= \text{Val}(P_v),\end{aligned}$$

which proves the lemma.

### A.2 Proof of Lemma 2

Assume towards contradiction that there exists a path  $P_v^* \in \mathcal{P}_{v,t}$  such that  $\text{Val}(P_v^*) < \text{Val}(P_v)$ . Let  $w \in U$  and  $z \in V \setminus U$  be two vertices such that  $zw \in P_v^*$ . Consider the partition of  $P_v^*$  given by  $P_v^* = P_v^*[v, z] \cup \{zw\} \cup P_v^*[w, t]$  (Note that if  $w = t$  then  $P_v^*[w, t] = \emptyset$  and if  $z = v$  then  $P_v^*[v, z] = \emptyset$ ).

By the choice of  $vu$  we have

$$\max\{\ell(vu) + y(u), s_v^{-vu}\} \leq \max\{\ell(zw) + y(w), s_z^{-zw}\},$$

which, by  $\text{Val}(P_u) = y(u)$  and  $\text{Val}(P_v^*[w, t]) \geq y(w)$  implies

$$\max\{\ell(vu) + \text{Val}(P_u), s_v^{-vu}\} \leq \max\{\ell(zw) + \text{Val}(P_v^*[w, t]), s_z^{-zw}\}.$$

The latter inequality and Lemma 1 give  $\text{Val}(P_v) \leq \text{Val}(P_v^*[z, t])$ . On the other hand, by monotonicity we have  $\text{Val}(P_v^*) \geq \text{Val}(P_v^*[z, t])$ . We conclude that  $\text{Val}(P_v) \leq \text{Val}(P_v^*[z, t]) \leq \text{Val}(P_v^*)$ ; a contradiction.

### A.3 Proof of Lemma 3

Clearly this choice represents a  $u$ - $t$  path in  $G - uu'$ , so  $s_u^{-uu'} \leq d_G(u, v) + \ell(vw) + d^*(w)$  holds. Assume towards contradiction that a better  $u$ - $t$  path  $P$  existed in  $G - uu'$ , namely  $\ell(P) < d_G(u, v) + \ell(vw) + d^*(w)$ . To reach  $t$  from  $u$  without using  $uu'$  the path  $P$  needs to contain an edge  $v'w'$ , such that  $v' \in T_u$  and  $w' \notin T_u$ . We obtain

$$\begin{aligned}\ell(P) &= \ell(P[u, v']) + \ell(v'w') + \ell(P[w', t]) \geq d_G[u, v'] + \ell(v'w') + d^*(w') \\ &\geq d_G(u, v) + \ell(vw) + d^*(w),\end{aligned}$$

which contradicts the choice of  $P$ .

## B A Summary of the Algorithm in Section 4

---

### Algorithm 2

---

```

1: Compute the shortest path tree  $T$ .
2: Create a copy  $\bar{T}$  of  $T$ .
3: Store a pointer  $p[u]$  in every vertex  $u$  in  $T$  to its corresponding vertex  $\bar{u}$  in  $\bar{T}$ .
4: Store a pointer  $p[\bar{u}]$  in every vertex  $\bar{u}$  in  $\bar{T}$  to its corresponding vertex  $u$  in  $T$ .
5: Store similar pointers  $p[\bar{e}]$  from edges in  $\bar{T}$  to corresponding edges in  $T$ .
6: Compute  $c_e$  for every  $e \in E'$ .
7: Sort  $E'$  according to increasing order of  $c_e$ . Let  $L$  be the sorted list.
8: while  $\bar{T}$  contains more than one vertex do
9:   Remove the first edge  $e = xy$  from  $L$ .
10:   $\bar{x} = p[x]$ ,  $\bar{y} = p[y]$ .
11:   $z = \text{lca}(e)$ ,  $\bar{z} = p[\text{lca}(e)]$ .
12:  Let  $Y \subset \bar{T}$  denote the union of the  $\bar{x}$ - $\bar{z}$  and  $\bar{y}$ - $\bar{z}$  paths in  $\bar{T}$ .
13:  for  $\bar{a} \in Y$  do
14:     $uu' = p[\bar{a}]$ .
15:     $s_u^{-uu'} = c_e - d^*(u)$ .
16:  Contract  $Y$  in  $\bar{T}$  and update pointers.
17:  repeat
18:    Let  $e = vw$  be the first edge in  $L$ .
19:    if  $p[v] = p[w]$  then
20:      Remove  $e$  from  $L$ .
21:  until  $p[v] \neq p[w]$  or  $L = \emptyset$ .
```

---

## C An Algorithm for $k$ -ORP

Algorithm 3 solves (4) for every  $v \in V - t$  and computes the corresponding optimal routing strategy. Note that Algorithm 3 is identical to Algorithm 1 when  $k = 1$ . To formally prove the correctness of Algorithm 3 one needs to state equivalent monotonicity properties, as well as analogues of Lemma 1 and Lemma 2. They are however omitted, as they are identical to those of Section 3.

We conclude by bounding the complexity of a naive implementation of this algorithm. Let  $T(m, n, k)$  denote the running time of the algorithm on a graph with  $n$  vertices and  $m$  edges and failure parameter  $k$ . The algorithm in Section 4 gives  $T(n, m, 1) = O(m \log n)$ . For  $k > 1$  we have  $T(m, n, k) = O(m + n \log n) + mT(m - 1, n, k - 1) = O(m^{k-1}T(n, m, 1))$ . This finally gives  $T(n, m, k) = O(m^k \log n)$ .

## D Pareto-Optimal ORPs

In this section we are concerned with obtaining a path  $P^*$  with robust length at most  $B$ , and a shortest nominal path length among all such paths. This problem

---

**Algorithm 3**

---

```
1: Compute  $y^{k-1}(u, E \setminus uv)$  for each  $uv \in E$ .
2:  $U = \emptyset$ .
3:  $W = V$ .
4:  $y^k(t, E) = 0$ .
5:  $y^k(u, E) = \infty$  for each  $u \in W$ .
6: while  $U \neq V$  do
7:   Find  $u = \arg \min_{z \in W} \{y^k(z, E)\}$ .
8:    $U = U + u$ .
9:    $W = W - u$ .
10: for all  $vu \in E$  such that  $v \in W$  do
11:   if  $y^k(v, E) > \max\{\ell(uv) + y^k(u, E), y^{k-1}(v, E \setminus \{vu\})\}$  then
12:      $y^k(v, E) = \max\{\ell(uv) + y^k(u, E), y^{k-1}(v, E \setminus \{vu\})\}$ .
```

---

is equivalent to that of finding a Pareto-optimal path with respect to the objective functions corresponding to the ordinary distance function, and the robust length. We assume  $B \geq OPT$ , the optimal solution value of the corresponding ORP instance. Formally, we aim at finding an  $s$ - $t$  path  $P^*$  satisfying

$$P^* = \arg \min_{P \in \mathcal{P}_{s,t}: \text{Val}(P) \leq B} l(P).$$

The algorithm for this problem is a slightly modified version of Dijkstra's algorithm on the directed edge-weighted graph  $G' = (V, K, \omega)$ , where  $K$  has two directed edges  $uv$  and  $vu$  for each undirected edge  $uv \in E$ , with  $\omega(uv) = \ell(uv) = \omega(vu)$ . Algorithm 4 is a formal statement of the algorithm. Note that the only difference with Dijkstra's algorithm is the condition in step 8. Unlike ordinary shortest paths, when a vertex  $u$  is selected, and the distance labels of its neighbors are updated, it is not sufficient to check for each  $uv \in K$  the usual condition

$$d(u) + \omega(uv) \leq d(v).$$

We need to additionally verify whether the edge  $uv$  can belong to a path with robust length  $B$  or not. In other words, we need to check whether the length of the path from  $s$  to  $u$  plus the length of the detour from  $u$  to  $t$  avoiding  $uv$  is at most  $B$ , namely

$$d(u) + s_u^{-uv} \leq B.$$

Algorithm 4 can clearly be implemented to run in  $O(m + n \log n)$  time using the results of Section 4.

Let us briefly discuss the potential applications of the latter problem. While shortest paths often have undesirable behavior in unreliable networks, an optimal solution to ORP might have a prohibitively large cost. In some applications faults occur rarely, hence it is preferred to have the cost of the nominal path as low as possible. At the same time, it is necessary that the cost does not exceed a certain threshold  $B$ , in every scenario. Consequently, it makes sense to regard the threshold  $B$  as a hard constraint on the robust length, and optimize the

---

**Algorithm 4**

---

```

1:  $S = \emptyset; \quad \bar{S} = V$ 
2:  $d(s) = 0; \quad d(v) = \infty \forall v \in V - s$ 
3: while  $t \notin S$  do
4:   Find  $u = \arg \min_{z \in \bar{S}} d(z)$ 
5:    $S = S + u$ 
6:    $\bar{S} = \bar{S} - u$ 
7:   for  $v \in N(u) \setminus S$  do
8:     if  $d(u) + \omega(uv) \leq d(v)$  and  $d(u) + s_u^{-uv} \leq B$  then
9:        $d(v) = d(u) + \omega(uv)$ 

```

---

length of the nominal path. Algorithm 4 gives the decision maker the desired freedom to choose the level of conservatism that she desires.

## E The Shortest Path Heuristics

It is common to use heuristics which rely on shortest paths in routing algorithms. In this section we show that a naive shortest path routing strategy performs very poorly for  $k$ -ORP. In fact, the approximation guarantee it provides grows exponentially with the adversarial budget  $k$ . To this end we define more formally the shortest path heuristics, which we denote by  $R_{SP}$ . The routing strategy  $R_{SP}$  works as follows. At each vertex  $u \in V$  and given a set of known failed edges  $F'$ ,  $R_{SP}$  tries to route the package along a shortest path in the remaining graph  $G - F'$ . In the following lemma we show that  $R_{SP}$  is a factor  $2^{k+1} - 1$  approximation for the optimal routing strategy, in the presence of at most  $k$  failed edges.

**Lemma 4.** *Let  $\mathcal{I} = (G, s, t)$  be an instance of  $k$ -ORP. Then*

$$\text{Val}_k(s, E, R_{SP}) \leq (2^{k+1} - 1)y^k(s, E).$$

*Proof.* Let  $OPT = y^k(s, E)$ ,  $R^* = \arg \min_R \text{Val}_k(s, E, R)$  and  $Q^0 = \theta_{R^*}(s, E, \emptyset)$  be the corresponding nominal path. Consider a set  $F$  of failed edges with  $|F| \leq k$ . Define  $\omega = \theta_{R_{SP}}(s, E, F)$  to be the walk followed by  $R_{SP}$  in  $G - F$ , and let  $\omega = (s = u_1, u_2, \dots, u_m = t)$  be the corresponding sequence of vertices. We divide our analysis according to the number of failed edges encountered by  $R_{SP}$ . We prove by induction on  $i$ , that if the routing strategy encountered a total of  $i \leq k$  failed edges then

$$\ell(\omega) \leq (2^{i+1} - 1)OPT. \quad (5)$$

The base case  $i = 0$  corresponds to scenarios in which no failed edge is encountered in the routing. In this case  $\omega$  is simply a shortest  $s$ - $t$  path in  $G$ , hence  $\ell(\omega) \leq OPT$ , as required. Assume next that (5) holds for every  $j < i$  and consider the case that the routing encounters exactly  $i$  failed edges. We can assume without loss of generality that all edges of  $F$  were probed and  $|F| = i$ .

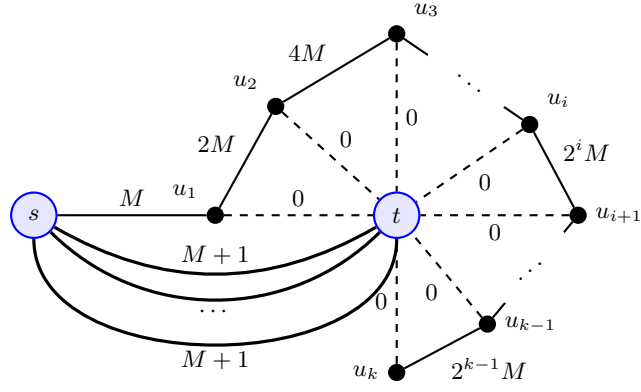


Let  $r < m$  be such that  $u_r$  is the vertex incident to the  $i$ 'th failed edge in the routing. In other words, before reaching  $u_r$ , the routing probed exactly  $i - 1$  failed edges. Let  $e = \{u_r, w\}$  be the  $i$ 'th failed edge probed by the routing strategy. Consider an execution of the routing strategy on the same instance with the different failure scenario  $F' = F - e$ . The resulting walk  $\omega'$  will have the first  $r$  vertices in common with  $\omega$ , namely the sub-walk  $\sigma = (s = u_1, \dots, u_r)$  will appear in both walks. By the inductive hypothesis we have that  $\ell(\sigma) \leq \ell(\omega') \leq (2^i - 1)OPT$ . It remains to bound the length of the tail of  $\omega$  from  $u_r$  until  $u_m = t$  to complete the proof. To this end recall that  $R_{SP}$  routes the package along the shortest remaining path in the graph. Since the last failure encountered by  $R_{SP}$  is  $e$ , the remaining path is simply the shortest  $u_r$ - $t$  path in  $G - F$ . To bound the length of this path we construct a  $u_r$ - $t$  walk  $\theta$  as follows. First  $\theta$  traces the entire route taken by  $R_{SP}$  back to  $s$  and then uses the walk that  $R^*$  would use to reach  $t$  from  $s$  in the scenario  $F$ . Clearly we have  $\ell(\theta) \leq (2^i - 1)OPT + OPT$ . Furthermore, this walk is intact in  $G - F$ . This gives the required bound  $\ell(\omega) \leq (2^i - 1)OPT + (2^i - 1)OPT + OPT = (2^{i+1} - 1)OPT$  and finishes the proof.  $\square$

The bound obtained in Lemma 4 seems crude at first glance. In particular, in the inductive step we follow the entire walk performed so far backwards to reach  $s$  and start over. In the following example we show that the bound of Lemma 4 is tight.

*Example 1.* Let  $M \in \mathbb{Z}_+$  be a large integer. Consider the following instance  $\mathcal{I} = (G, s, t)$  of  $k$ -ORP. The graph contains  $k + 1$  parallel edges connecting  $s$  and  $t$  with length  $M + 1$ . In addition the graph contains a path  $(s, u_1, \dots, u_k)$  of length  $k + 1$ . The edge  $su_1$  has length  $M$  and every edge  $u_i u_{i+1}$  has length  $2^i M$ . Finally, the vertices  $u_1, \dots, u_k$  are connected to  $t$  with edges of length zero. The construction is illustrated in Figure 2.

Consider the failure scenario, which fails all edges  $u_i t$  for  $i \in [k]$ . The routing strategy  $R_{SP}$  will follow the path  $(s, u_1, \dots, u_k)$ , then follow it back to  $s$  and then take one of the edges with length  $M + 1$  to  $t$ . The total length of this walk is  $2(M + 2M + \dots + 2^k M) + M + 1 = (2^{k+1} - 1)M + 1$ . At the same time the optimal routing strategy routes the package along the edges with length  $M + 1$  with a worst-case cost of  $M + 1$ . The ratio between the two numbers tends to  $2^{k+1} - 1$  as  $M$  tends to infinity.



**Fig. 2.** A bad example for the shortest path heuristic. The dashed edges correspond to the worst case scenario.